AMENDMENTS TO THE SPECIFICATION

Please replace the Title on page 1 with the following amended Title:

EFFICIENT EVENT NOTIFICATION IN [[A]] CLUSTERED COMPUTING ENVIRONMENTS

Please replace paragraph [0020] with the following amended paragraph:

[0020] Returning to step 104, if it is [[was]] determined that the event was previously generated, method 100 proceeds to step 105. At step 105, a determination is made whether the event was previously propagated. If the event was not propagated, an identifier is set at the source server node indicating that the event was previously generated (as determined in step 104) and not propagated (as determined in step 105). Before coalescing multiple event notifications to a single event notification, it is desirable to determine that the event has been previously generated (indicating that there is a possibility of sending multiple event notifications) and that the previously generated event has not yet been propagated (indicating that avoiding the sending of redundant event notifications and by coalescing of events is possible). If in step 105 it is determined that the event has already been propagated, method 100 proceeds to step 107. At step 107 an identifier is set indicating that the event was previously generated and was propagated, and the new event cannot be coalesced. Method 100 then proceeds to step 112, where it is appended onto an existing message for propagation in step 114. If in step 105 it is determined that the previously generated event has not yet been propagated, method 100 proceeds to step 108. At step 108, an identifier is set at the source server node indicating that multiple "same" event notifications with the identical event ID and parameter information exist (step 108). In an embodiment, same event notifications are two notifications of the same occurrence of the same

event. In an embodiment, same event notifications are notification of different occurrences of the same event. Next, for event notification efficiency, multiple event notifications are coalesced into a single event notification, as depicted at step 110. Next, at step 112, the coalesced event notification can then be propagated onto the subscriber nodes of the clustered computing environment. Also, in step 112 the event notification is then appended onto an existing message, and method 100 continues with step 114, as described above.

Please replace paragraph [0029] with the following amended paragraph:

[0029] In FIG. 3B, a first event 302, having event ID#5, is generated at the local node, and has not yet been appended and propagated to any of the remote subscriber nodes. In FIG. 3B, event 302 has not yet been propagated to any of the subscriber nodes[[,]]; therefore all bits in bit vector 380 are set to a first value (e.g., "1") for nodes 342, 343, and 344. Tail pointer 301 moves to accommodate bit vector 380, thereby indicating the presence of a notification in circular buffer 300 that needs to still be sent to each of nodes 342, 343, and 344, respectively. Each new bit vector is added to the tail of the circular buffer 300. Event 302 (with ID #5) is then propagated by the local node to a single remote subscriber node, which is node 344. Event 302 has not yet been transmitted to nodes 342 and 343. Therefore, the bits in bit vector 380 remain in their first value (e.g., "1") for nodes 342 and 343, and is changed to a second value (e.g., "0") for node 344. (In the remainder of this example, "1" will be used for the first value, and "0" will be used for the second value. However, the first and second values may be any other pair of two different values.) As shown in FIG. 3C, the head and tail pointer for node 344 is repositioned accordingly. Specifically, head pointers for nodes 342 and 343 (h342 and h343) remain at their starting position, and the head pointer for node 344 (h344), now the fastest head pointer, moves to the

same position as tail pointer 301 to indicate that no notification needs to be sent to node 344. A second unique event is generated by the local node, which is depicted in FIG. 3D as event 304 with event ID #6. Event 304 has not been propagated to any of the subscriber nodes, therefore all the bits in bit vector 382 are set to "1". The tail pointer 301 advances accordingly. Since event 304 has not yet been propagated to any of the subscriber nodes (342, 343, and 344), the head pointers (h342, h343, and h344) for the subscriber nodes remain at their respective positions.

Please replace paragraph [0030] with the following amended paragraph:

[0030] In FIG. 3E, to illustrate the introduction of a same event (having event ID#5) that has been previously generated and not yet fully propagated to all subscriber nodes, a third event, event 306, is generated which is identical to a previously generated event, event 302. Specifically, event 306 is the second event that has event ID #5 that was generated by the local node. During a sweep of circular buffer 300, previously generated event 302 with event ID #5 has already been propagated to node 344, but has not been propagated to nodes 343 and 34[[3]]2 as indicted by the "0" in the bit of bit vector 380 corresponding to node 344 and the "1"s in the bits of bit vector 380 corresponding to nodes 343 and 344-. Therefore, the bits in bit vector 384 are set to "1" for node 344, and set to "0" for nodes 343 and 342, indicating not to send event 306 to nodes 342 and 343 -and to send event 306 to node 344. Since the earlier event (event 302) with event ID #5 is still waiting to be, and presumably will be, propagated to nodes 343 and 342, therefore event 306 does not need to be sent to nodes 342 and 343.

Please replace paragraph [0033] with the following amended paragraph:

[0033] FIG. 3G[[,]] illustrates a third generation of an event, event ID #5, [[that]] which has been previously generated as events 302 and 306 for another node in the cluster. A sweep of circular buffer 300 reveals that between the fastest head pointer, head pointer h342, and tail pointer 301 there already exists an entry in circular buffer 300 (event 306) that has event ID #5. Consequently, rather than creating a third entry for event ID #5, the last entry for event ID #5, event 306, is modified. Therefore, as illustrated at bit vector 384, the bit corresponding to node 342 is set to "1", indicating that event 306 with ID #5 has not yet been propagated to node 342. The bit in bit vector 384 corresponding to node 344 remains at "1", indicating that event 306 has not been propagated to node 344. Accordingly, the head pointers for nodes 342, 343, and 344 remain at their previous positions. FIG. 3H illustrates a previously generated event, event 304 having event ID #6, being appended onto an existing message and propagated to a single node, node 342. Since the event 304 is propagated to node 342, therefore the head pointer h342 for node 342 is advanced accordingly and the bit corresponding to node 342 in bit vector 382 is set to "0" indicating that event 304 has been propagated to node 342.

Please replace paragraph [0045] with the following amended paragraph:

[0045] Another alternative would be to employ a fixed-size messaging technique. For example, a buffer size could be fixed. According to an aspect or embodiment, since the buffer size would be fixed, the free space would be pre-determined and events would be coalesced and event notifications could be appended onto existing message traffic until the free space is filled. Furthermore, many notifications could be appended onto a message, and the procedure would be repeated indefinitely. For example, if a message is about to be sent to a particular node, all the event notifications that have not been sent to that node or that need to be sent to that node could

be appended to that message. Yet another variation is the employment of a priority queuing mechanism in order to determine priority for events. High priority events would supercede low priority events in the notification queue. Thus, if for example limited space is available for appending event notifications to a message that is ready to be sent, the high priority event notifications would be appended before the low priority notifications. In some embodiments, the ~~priority~~ priorities of certain types or of all types of event notifications may be increased according to how long it has been waiting and/or what other processes and/or events have occurred or are in process. In some embodiments, high priority event notifications may not wait for a message to be ready and are sent without piggy backing. In some embodiments, high priority and/or low priority messages may be sent without piggy backing after they have been waiting beyond a certain amount of time.